

```

-- ccmv3
-- version5 - 2/18/2004 - modified to accept 2 FPGA design
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
library synplify;
use synplify.attributes.all;

entity ccmv3 is port

(clock          : in std_logic;
resetn         : in std_logic;
busy           : in std_logic;
done            : in std_logic;
brd_ad         : in std_logic_vector(7 downto 0);
brcst          : in std_logic_vector(5 downto 2);

-- SCC signals
parity_error   : in std_logic;
overflow        : in std_logic;
TXrdy          : in std_logic; -- serial data can be sent to UART, TXrdy
receive_full   : in std_logic; -- serial data_in available from UART, receive_full
SCC_Data_RCVD : in std_logic_vector(7 downto 0); -- SCC_SCC_Data_RCVD[7..0]

WEn            : out std_logic; -- load serial data_out into UART, WEn
OEn            : out std_logic; -- registered serial data_in from UART, OEn
SCC_Data_TRXD : out std_logic_vector(7 downto 0); -- SCC_data_in[7..0]
state_out      : out std_logic_vector(4 downto 0);

-- I2C Control signals
Control_Register       : out std_logic_vector(6 downto 0);
Number_Transfers_Register : out std_logic_vector(3 downto 0);
Pointer_Register        : out std_logic_vector(7 downto 0);
Data_Register           : out std_logic_vector(7 downto 0);
operate                : out std_logic; -- operate on I2C or FRAM
R_W_control             : out std_logic;

-- internal RAM signals
RAM_ADDRESS         : out std_logic_vector(4 downto 0);
RAM_DATA_D          : out std_logic_vector(7 downto 0);
RAM_WRITE_ENABLE    : out std_logic;
RAM_READ_ENABLE     : out std_logic;
RAM_DATA_Q          : in  std_logic_vector(7 downto 0);

-- power trip signals
pwr_trip1           : in std_logic;
pwr_trip2           : in std_logic;
error_reported      : in std_logic;

QPLLA_LOCK          : in std_logic_vector(7 downto 0);
QPLL_B_LOCK         : in std_logic_vector(7 downto 0);

-- Fast download signal
fram_download        : out std_logic;

-- A to D control signal
AD_RUN              : out std_logic;
-- reset signals
FE_reset             : out std_logic;
FE_reset_PLL         : out std_logic;
CCM_reset            : out std_logic;
TTC_reset            : out std_logic;
Clock_select         : out std_logic;
QIE_Reset            : out std_logic; -- used to be BZERO_out
--Delay_register       : out std_logic_vector(7 downto 0);
Power_reset          : out std_logic);

```

```

end ccmv3;

architecture behave_rtl of ccmv3 is
attribute syn_radhardlevel of behave_rtl : architecture is "tmr";
--attribute syn_enum_encoding of behave_rtl : architecture is "sequential";

type state_values is (st0, st1, st2, st3, st4, st5, st6, st7, st8, st9, st10, st11, st12,
st13, st14, st15, st16, st17, st18, st19, st20, st21);
signal pres_state          : state_values;
attribute syn_state_machine : boolean;
attribute syn_state_machine of pres_state : signal is false;
signal register_add_pointer : std_logic_vector(4 downto 0);
signal array_size           : std_logic_vector(3 downto 0);
signal STATUS_REGISTER_63   : std_logic_vector(7 downto 0);

signal buff_buff            : std_logic_vector(7 downto 0);
signal read_temp             : std_logic;
signal load_timer            : std_logic;
signal timer_en              : std_logic;
signal timed_out             : std_logic;
signal timer                 : std_logic_vector(12 downto 0);

signal CCM_reset_enable      : std_logic;
signal QIE_Reset_SC          : std_logic;
signal PLL_Reset_SC          : std_logic;
signal Delay_register_enable  : std_logic;
signal Delay_register         : std_logic_vector(7 downto 0);
signal Broadcast_register_enable : std_logic;
signal Broadcast_register     : std_logic_vector(3 downto 0);
signal QIE_Reset_TEMP         : std_logic;
signal QIE_Reset_ENABLE        : std_logic;
signal Delay                  : std_logic_vector(7 downto 0);
signal READ_QPLLA             : std_logic;
signal READ_QPLL_B            : std_logic;

--constant busy      : std_logic := '0';
--constant done       : std_logic := '0';

begin
  -- fsm register
  state_reg: process (clock, resetn)
    begin
      if (resetn = '0') then
        pres_state      <= st0;
        SCC_Data_TRXD  <= "00000000";
        RAM_ADDRESS     <= "0000";
        read_temp        <= '0';
        CCM_reset_enable <= '0';
        CCM_reset        <= '0';
        FE_reset         <= '1';
        Power_reset      <= '0'; -- reset in the other regulator reset block
        TTC_reset         <= '0'; -- changed to 0 on 2/18/04
        Clock_select     <= '0'; -- modify for simualtion '1' = download on res
      et
        AD_RUN          <= '0';
        fram_download   <= '0'; -- modify for simualtion '1' = download on res
      et
        operate          <= '0';
        R_W_control      <= '0';
        WEn              <= '1';
        OEn              <= '1';
        RAM_READ_ENABLE   <= '0';
        RAM_WRITE_ENABLE  <= '0';
        RAM_DATA_D        <= "00000000";
      end if;
    end process state_reg;
  end;

```

```

register_add_pointer <= "00000";
array_size           <= "0000";

Control_Register      <= "0000000";
Number_Transfers_Register <= "0000";
Pointer_Register     <= "00000000";
Data_Register         <= "00000000";
load_timer <= '0';
timed_out <= '0';
timer_en <= '0';
timer <= "10010110000000";
STATUS_REGISTER_63 <= "00000000";

Broadcast_register   <= "0000";
Delay_register        <= "00000001";
Delay_register_enable <= '0';
Broadcast_register_enable <= '0';
FE_reset_PLL          <= '1';
QIE_Reset             <= '0'; -- BZERO_out
QIE_Reset_TEMP         <= '0';
QIE_Reset_ENABLE       <= '0';
Delay                 <= "00000000";
PLL_Reset_SC          <= '0';
QIE_Reset_SC          <= '0';
READ_QPLLA            <= '0';
READ_QPLLB            <= '0';

elsif clock'event AND clock = '1' then
-----
FE_Reset_PLL <= PLL_Reset_SC or (brcst(4) and Broadcast_register(2)); -- sc stands for slow controls
QIE_Reset_TEMP <= QIE_Reset_SC or (brcst(2) and Broadcast_register(0));

if QIE_Reset_TEMP = '1' then
  QIE_Reset_ENABLE <= '1';
end if;

if QIE_Reset_ENABLE = '1' then -- counter to count up to delay register value
  Delay <= Delay + 1;           -- to set the delay before a QIE_Reset is issued
else
  Delay <= "00000000";
end if;

if Delay = Delay_register then
  QIE_Reset <= '1'; -- BZERO_out
  QIE_Reset_ENABLE <= '0';
else
  QIE_Reset <= '0';
end if;
-----
-- set status register bits
  STATUS_REGISTER_63(0) <= busy;

  if busy = '1' and done ='1' then
    fram_download <= '0';
    operate <= '0';
    STATUS_REGISTER_63(1) <= '0';
  end if;

  If Clock_select = '1' then
    STATUS_REGISTER_63(2) <= '1';
  end if;

  If error_reported = '1' then
    STATUS_REGISTER_63(3) <= '1';
  end if;

```

```

        If pwr_trip1 = '1' then
            STATUS_REGISTER_63(4) <= '1';
        end if;

        If pwr_trip2 = '1' then
            STATUS_REGISTER_63(5) <= '1';
        end if;

-----
        if load_timer = '1' then
            timer <= "1001011000000"; -- change to > 104us length of 2nd byte of trans
fer
        elsif timer_en = '1' then -- this number represents 120us
            timer <= timer - 1;
        else
            timer <= timer;
        end if;

        if timer = "00000000000000" then
            timed_out <= '1';
        else
            timed_out <= '0';
        end if;
-----
case pres_state is
    when st0 =>
        state_out <= ("00000");
        WEn <= '1';
        OEn <= '1';
        RAM_READ_ENABLE      <= '0';
        RAM_WRITE_ENABLE     <= '0';
        CCM_reset            <= '0';
        FE_reset             <= '0';
        PLL_Reset_SC         <= '0';
        TTC_reset            <= '0';
        CCM_reset_enable     <= '0';
        Delay_register_enable <= '0';
        Broadcast_register_enable <= '0';
        --
        fram_download <= '0';
        QIE_Reset_SC         <= '0';
        timer_en             <= '0';
        read_temp             <= '0';
        Power_reset           <= '0';

        if receive_full = '1' then
            OEn <= '0'; -- pass data from SCC to CCM
            pres_state <= st21;
        else -- receive_full = 0 so stay at st0
            if busy = '1' then
                AD_RUN <= '0';
            else
                AD_RUN <= '1';
            end if;
            register_add_pointer <= (others => '0');
            pres_state <= st0;
        end if;

    when st21 =>
        state_out          <= ("10101");
        register_add_pointer <= SCC_Data_RCVD(4 downto 0);
        STATUS_REGISTER_63(7) <= parity_error;
        STATUS_REGISTER_63(6) <= overflow;
        AD_RUN <= '0';
        if busy = '0' then
            case SCC_Data_RCVD(7 downto 6) is
                when "01"    => pres_state <= st1; -- read register

```

```

        when "10"    => pres_state <= st2; -- write register
        when "11"    => pres_state <= st6;   -- array write registers
        when others => pres_state <= st20; load_timer <= '1'; --load_t
imer <= '1';
            end case;
        else
            pres_state <= st19; -- return board address and status - busy
        end if;

-- read sequence
    when st1 => -- return status
        state_out      <= ("00001");
        read_temp      <= '1';
        OEn           <= '1'; -- reset receiver signals
        case register_add_pointer is
            when "11010" => RAM_READ_ENABLE <= '0'; READ_QPLLA <= '1'; RE
AD_QPLLB <= '0';-- read QPLLA register 26
            when "11011" => RAM_READ_ENABLE <= '0'; READ_QPLLA <= '0'; RE
AD_QPLLB <= '1';-- read QPLLA register 27
            when others   => RAM_READ_ENABLE <= '1'; READ_QPLLA <= '0'; RE
AD_QPLLB <= '0';--read RAM data
        end case;
        RAM_ADDRESS   <= register_add_pointer;
        pres_state <= st19;

-- Write single byte sequence
    when st2 => -- enable the RAM address
        state_out <= ("00010");
        if SCC_Data_RCVD(5 downto 0) = "111111" then -- address 63/31
            CCM_reset_enable <= '1';
            Delay_register_enable <= '0';
            Broadcast_register_enable <= '0';
        elsif SCC_Data_RCVD(5 downto 0) = "111110" then -- address 62/30
            CCM_reset_enable <= '0';
            Delay_register_enable <= '1';
            Broadcast_register_enable <= '0';
        elsif SCC_Data_RCVD(5 downto 0) = "111101" then -- address 61/29
            CCM_reset_enable <= '0';
            Delay_register_enable <= '0';
            Broadcast_register_enable <= '1';
        else
            CCM_reset_enable <= '0';
            Delay_register_enable <= '0';
        end if;

        RAM_ADDRESS <= register_add_pointer;
        OEn <= '1'; -- reset the receiver buffer
        load_timer <= '1';
        pres_state <= st3;

when st3 => -- OEn needs to be reset
        state_out <= ("00011");
        load_timer <= '0';
        if (receive_full = '1') then
            OEn <= '0';
            timer_en <= '0';
            pres_state <= st4;
        else
            timer_en <= '1';
            if timed_out = '1' then
                pres_state <= st0;
            else
                pres_state <= st3;
            end if;
        end if;

when st4 => -- write value to address
        state_out <= ("00100");

```

```

        if CCM_reset_enable = '1' then
            FE_reset           <= SCC_Data_RCVD(7);
            PLL_Reset_SC      <= SCC_Data_RCVD(6);
            CCM_reset          <= SCC_Data_RCVD(5);
            TTC_reset          <= SCC_Data_RCVD(4);
            Clock_select       <= SCC_Data_RCVD(3);
            QIE_RESET_SC       <= SCC_Data_RCVD(2);
            fram_download      <= SCC_Data_RCVD(1);
            Power_reset         <= SCC_Data_RCVD(0);
        end if;
        if Delay_register_enable = '1' then
            Delay_register <= SCC_Data_RCVD;
        end if;
        if Broadcast_register_enable = '1' then
            Broadcast_register(3 downto 0) <= SCC_Data_RCVD(3 downto 0);

        end if;
        if (receive_full = '0') then
            OEn <= '1';           -- reset receiver signals
            RAM_DATA_D <= SCC_Data_RCVD;
            RAM_WRITE_ENABLE <= '1';
            pres_state <= st5;
        else
            pres_state <= st4;
        end if;

when st5 => -- register status register bits
    state_out <= ("00101");
    QIE_RESET_SC      <= '0';
    TTC_reset          <= '0';
    CCM_reset_enable <= '0';
    Delay_register_enable <= '0';
    Broadcast_register_enable <= '0';
    pres_state         <= st19;

-- Write array of bytes sequence
when st6 => -- enable the RAM address
    state_out <= ("00110");
    RAM_ADDRESS <= register_add_pointer;
    OEN           <= '1';           -- enable the receiver buffer
    load_timer     <= '1';
    pres_state     <= st7;

when st7 => -- OEN needs to be reset
    state_out <= ("00111");
    load_timer <= '0';
    if (receive_full = '1') then
        OEn           <= '0'; -- reset the receiver buffer
        timer_en      <= '0';
        pres_state    <= st8;
    else
        timer_en <= '1';
        if timed_out = '1' then
            pres_state <= st0;
        else
            pres_state <= st7;
        end if;
    end if;

when st8 => -- write value to array_size
    state_out <= ("01000");
    if (receive_full = '0') then
        OEn <= '1';           -- reset the receiver buffer
        array_size <= SCC_Data_RCVD(3 downto 0);
        load_timer <= '1';
        pres_state <= st9;
    else

```

```

        pres_state <= st8;
    end if;

when st9 => -- OEn needs to be reset
    state_out <= ("01001");
    load_timer <= '0';
    if (receive_full = '1') then
        OEn <= '0';          -- reset the receiver buffer
        timer_en     <= '0';
        pres_state   <= st10;
    else
        timer_en     <= '1';
        if timed_out = '1' then
            pres_state <= st0;
        else
            pres_state <= st9;
        end if;
    end if;

when st10 => -- write next byte to address(0)
    state_out <= ("01010");
    if (receive_full = '0') then
        OEn                  <= '1';
        RAM_DATA_D           <= SCC_Data_RCVD;
        RAM_WRITE_ENABLE      <= '1';
        load_timer            <= '1';
        Case register_add_pointer is
            when "00000" => control_register           <= SCC_data_RCV
D(6 downto 0); R_W_control <= SCC_data_RCVD(7);
            when "00001" => number_transfers_register <= SCC_data_RCV
D(3 downto 0);
            when "00010" => pointer_register           <= SCC_data_RCV
D;
            when "00011" => data_register              <= SCC_data_RCV
D;
            when others => buff_buff                 <= SCC_data_RC
VD;
        end case;
        pres_state <= st11;
    else
        pres_state <= st10;
    end if;

when st11 => --
    state_out <= ("01011");
    RAM_WRITE_ENABLE <= '0';
    load_timer <= '0';
    if array_size = 1 then
        operate      <= '1'; -- send command to start I2C or FRAM operatio
n
        pres_state   <= st19; -- return board address
    else
        register_add_pointer <= register_add_pointer + 1;
        pres_state   <= st12; -- do another write
    end if;

when st12 => -- Increment RAM address and do next write
    state_out <= ("01100");
    array_size   <= array_size - 1;
    RAM_ADDRESS <= register_add_pointer;
    pres_state   <= st9;

when st19 => -- return board address
    state_out <= ("10011");
    pres_state <= st13;

```

```

when st13 => -- return board address
    state_out <= ("01101");
    --CCM_reset_enable <= '0';
    if (busy = '0' and TXrdy = '1') then
        SCC_Data_TRXD <= brd_ad;
        WEn <= '0'; -- register Board Address in the UART to transmit

        pres_state <= st14;
    else
        pres_state <= st13;
    end if;

when st14 => -- WEn needs to be reset
    state_out <= ("01110");
    if (TXrdy = '0') then -- wait until it goes low
        pres_state <= st15;
        WEn <= '1'; -- set back to normal
    else
        pres_state <= st14;
    end if;

when st15 => -- return status
    state_out <= ("01111");
    if (TXrdy = '1') then
        SCC_Data_TRXD <= STATUS_REGISTER_63;
        WEn <= '0'; -- register status data in the UART to transmit

        pres_state <= st16;
    else
        pres_state <= st15;
    end if;

when st16 => -- WEn needs to be reset
    state_out <= ("10000");
    if (TXrdy = '0') then -- wait until it goes low
        WEn <= '1';
        STATUS_REGISTER_63 <= "00000000";
        if read_temp = '1' then
            pres_state <= st17;
        else
            pres_state <= st0;
        end if;
        -- set back to normal
    else
        pres_state <= st16;
    end if;

when st17 => -- return data from register being pointed to
    state_out <= ("10001");
    if (TXrdy = '1') then
        if (READ_QPLLA = '1') then
            SCC_Data_TRXD <= QPLLA_LOCK;
        elsif (READ_QPLLB = '1') then
            SCC_Data_TRXD <= QPLLB_LOCK;
        else
            SCC_Data_TRXD <= RAM_DATA_Q;
        end if;
        WEn <= '0'; -- register RAM data in the UART to transmit

        pres_state <= st18;
    else
        pres_state <= st17;
    end if;

when st18 => --
    state_out <= ("10010");
    if (TXrdy = '0') then

```

```
        WEn          <= '1'; -- reset wen
        pres_state  <= st0;
    else
        pres_state  <= st18;
    end if;

when st20 => -- wait for 11 bits to transfer
    state_out <= ("10100");
    timer_en <= '1';
    load_timer <= '0';
    OEn          <= '1'; -- reset receiver signals
    if timed_out = '1' then --
        pres_state <= st0; --
    else
        pres_state <= st20;
    end if;

end case;
end if;
end process;
end behave_rtl;
```